爱创课堂前端培训

JS 基础

第 8 天课堂笔记（本课程共 8 天）

班级：北京前端训练营 7 期

日期：2017 年 5 月 20 日

爱创课堂官网 ： www.icketang.com

目录

复习

获取 DOM 元素：

document.getElementById(“id”)。

id 或者 name 是唯一。

html 操作：

点方法或者 getAttribute（）seAttribute()

可以对我们 html 元素或者标签进行读取或者设置。

getAttribute（）seAttribute()只用于自定义属性的获取或者设置。其他的都用点方法。

css 操作：

目前学习的方法的都是只能对行内样式进行读取或者是设置。

```
box.style.backgroundColor = "pink";
```

```
<div id="box" style="background-color: pink;"></div> == $0
```

事件：当元素触发该事件时，会立即执行函数。

一、getElementsByTagName()

1.1 概述

getElementsByTagName()和 getElementById（）他们是全线兼容。get 获得 Elements 元素们 By 通过 Tag 标签 Name 名。

getElementsByClassName 通过类名获取元素，有严重的兼容性问题。

通过标签名获得的对象是一个装有所有 p 标签对象的数组。

```
var ps = document.getElementsByTagName("p");
console.log(ps);
console.log(typeof ps);
```

```
▼HTMLCollection[7]
  ▶0: p
  ▶1: p
  ▶2: p
  ▶3: p
  ▶4: p
  ▶5: p
  ▶6: p
    length: 7
  ▶__proto__: HTMLCollection
object
```

数组的每一项可以通过下标获取，得到的每一项数据也都是一个对象类型，本身具有与 id 类似属性和方法。

```
for(var i = 0 ; i < ps.length ; i ++){
    console.log(ps[i].innerHTML);
}
```

```
1
2
3
4
5
6
7
```

通过标签获取元素，不在乎元素的嵌套的多深，都能够被找到。

数组里元素的顺序与嵌套关系无关，与标签的首次出现位置有关，出现在前面，在数组的位置就靠前。

```
<div class="box1" style="background-color:#eee;">
    <div class="box2" style="background-color:#f00;"></div>
    <div class="box3" style="background-color:#ff0;">
        <div class="box4" style="background-color:#fff;"></div>
```

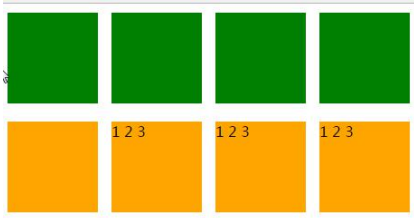
```
        <div class="box5" style="background-color:#0f0;"></div>
    </div>
</div>
<div class="box6" style="background-color:#00f;"></div>
```

```
rgb(238, 238, 238)
rgb(255, 0, 0)
rgb(255, 255, 0)
rgb(255, 255, 255)
rgb(0, 255, 0)
rgb(0, 0, 255)
```

1.2 连续打点调用

不但 document 可以使用 get 方法，任何 html 元素也可以使用。并且可以连续打点调用。

```
// 得到 box1 元素,对象可以打点调用 get
var box1 = document.getElementById("box1");
// 得到 box1 里面的所有标签 p
var ps1 = box1.getElementsByTagName("p");
// 得到 box2 里面的 所有 p 元素
var ps2 = document.getElementById("box2").getElementsByTagName("p");
for(var i = 0 ; i < ps1.length ; i ++){
    ps1[i].style.backgroundColor = "green";
}
// 遍历
for(var i = 0 ; i < ps2.length ; i ++){
    ps2[i].style.backgroundColor = "orange";
}
```



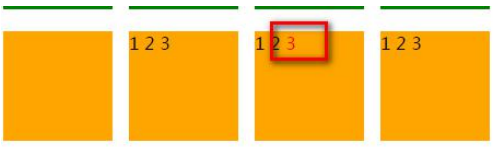
错误写法：

```
var ps2 = document.getElementById("box2").document.getElementsByTagName("p");
```

getElementsByTagName()也可以连续打点。

同过标签得到下标为 1 的 div,然后下标 2 得到 p 再然后通过下标 2 得到 span 让这个 span 颜色变红。

```
var spans2 = document.getElementsByTagName("div")[1].getElementsByTagName("p")[2].getElementsByTagName("span")[2];
spans2.style.color = "red";
```



虽然 getElementsByTagName 可以连续打点得到任何我想要的元素，但是过程有点繁琐。

实际工作中我们习惯 id 和 tagName 配合使用。

即使通过标签得到元素只有唯一的一个，也需要加下标得到该元素。

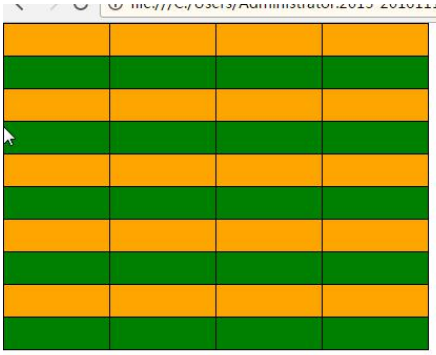
```
var op = document.getElementById("box3").getElementsByTagName("p");
op.style.backgroundColor = "yellowgreen";
```

```
Uncaught TypeError: Cannot set property 'backgroundColor' of undefined
    at window.onload (06_连续技点.html:52)
```

```
var op = document.getElementById("box3").getElementsByTagName("p")[0];
op.style.backgroundColor = "yellowgreen";
```

案例：表格的隔行变色

```
// 得到元素
var trs = document.getElementsByTagName("tr");
// 奇数行
for(var i = 1 ; i < trs.length ; i += 2){
    trs[i].style.backgroundColor = "green";
}
// 偶数行
for(var i = 0 ; i < trs.length ; i += 2){
    trs[i].style.backgroundColor = "orange";
}
```



1.3 批量添加事件

```
var ps = document.getElementById("box1").getElementsByTagName("p");
// 数组遍历
for(var i = 0 ; i < ps.length ; i ++){
    ps[i].onclick = function(){
    console.log(i);
    }
}
```

由于闭包的影响，我们的 i 注意变化的量。循环完毕之后 i 变为 4，输出 i = 4。

解决办法：IIFE

```
// IIFE 解决
for(var i = 0 ; i < ps.length ; i ++){
    (function(a){
    ps[a].onclick = function(){
    console.log(ps[a].innerHTML);
    }
   })(i);
}
```

还可以使用函数内部的关键字 this。“这个”。

this 只能用于函数的内部，指代触发这个事件的对象。包括这个对象的所有方法和属性。

```
for(var i = 0 ; i < ps.length ; i ++){
    ps[i].onclick = function(){
    console.log(this.innerHTML);
    }
}
```


```
for(var i = 0 ; i < ps.length ; i ++){
// 给每一个 p 自定义一个属性 index
    ps[i].index = i;
// ps[0].index = 0;
// ps[1].index = 1;
// ps[2].index = 2;
// ps[3].index = 3;

ps[i].onclick = function(){
    // console.log(i);
    console.log(this.index);
    }
}
```

案例：批量控制全选或者全不选。

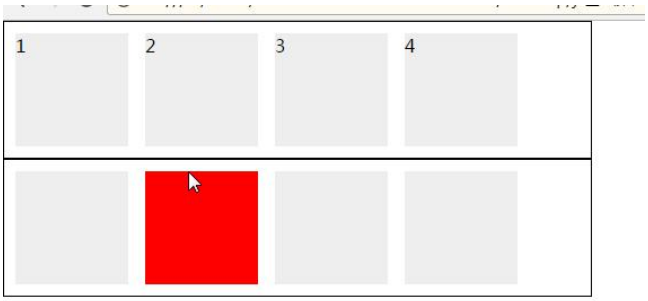
```
var ipts = document.getElementsByTagName("input");
var btn = document.getElementById("btn");
// 全选
btn.onclick = function(){
    if(btn.value == "全选"){
        // 全选时，需要做两件事
        // 批量添加选中状态
        for(var i = 0 ; i < ipts.length ; i ++){
            ipts[i].checked = "checked";
```

```
        }
        // 第二件事全选变为全不选
        btn.value = "全不选";
    }else{// 全不选
        for(var i = 0 ; i < ipts.length ; i ++){
            ipts[i].checked = "";
        }
        btn.value = "全选";
    }
```

1.4 对应思想

我们喜欢通过控制一个元素，去改变另外一个对应的元素。在写程序的时候，我们习惯了找到他们之间的一个联系，通过它对应控制。

```
var ps1 = document.getElementById("box1").getElementsByTagName("p");
var ps2 = document.getElementById("box2").getElementsByTagName("p");
// 给 ps1 批量添加事件
for(var i = 0 ; i < ps1.length ; i ++){
    // 把 i 的值存入我们的 index 属性
    ps1[i].index = i;
    ps1[i].onclick = function(){
        ps2[this.index].style.backgroundColor = "red";
    }
}
```



1.5 排他思想

保留自己，排除别人。

多用于批量控制，有一个与其他不同，先整体设置一个共同的样式，再给需要变化的元素单独加特殊样式。

```
// 给 ps1 批量添加事件
for(var i = 0 ; i < ps1.length ; i ++){
    // 把 i 的值存入我们的 index 属性
    ps1[i].index = i;
    // 排他思想
    ps1[i].onclick = function(){
        // 批量控制 ps2 都为#eee
        for(var j = 0 ; j < ps2.length ; j ++){
            ps2[j].style.backgroundColor = "#eee";
        }
        // 对应的下标再变红
        ps2[this.index].style.backgroundColor = "red";
        }

}
```

二、计算后的样式

2.1 高级浏览器

计算后的样式：指的是 HTML 元素在 css 各种选择器综合作用下，得到的最终样式。

W3C 制定的标准 API，所有现代浏览器（包括 IE9，但不包括之前的版本）都实现了 window.getComputedStyle()，该方法接收一个要进行样式计算的元素，并返回一个可以进行属性查询的接口。返回接口提供了一个名为 getPropertyValue()的方法，用于检索特定样式属性的计算样式。getPropertyValue 方法接收 css 属性名称，而不是驼峰式的名称。getPropertyValue()可以不写，直接用方括号来检索属性也可以。

getPropertyValue：get 获得 property 属性 value 值，得到的是具体的某一个属性的值。

```
var image = document.getElementById("image");
console.log(image.style.borderRightColor);   //点语法不能得到计算后的样式
console.log(window.getComputedStyle(image).getPropertyValue("left"));
```

getComputedStyle 这个方法是 window 对象的方法。是 BOM（browser object model 浏览器对象模型）里面的一个最重要的对象。

之前学习的 alert（）就是 window 对象的方法。习惯不写 window。

```
window.alert("哈哈，我是 window 的对象");
```

getComputedStyle 正确调用方法：

调用时内部的对象就用变量，不要加引号。

```
window.getComputedStyle(对象);
```

注意：这个方法有兼容问题。适用于高级浏览器和 IE9 以上的。

getPropertyValue 注意：里面的属性名字需要写在一对小括号里，名字 css 里面怎么写，直接用。不用写成驼峰式。

```
box.innerHTML = window.getComputedStyle(img).getPropertyValue("borderRightColor"); //错误的
```

```
box.innerHTML = window.getComputedStyle(img).getPropertyValue("border-right-color");
//正确的
```

注意：不管怎么书写，属性名都需要加双引号。[]形式可以使用-形式，也可使用驼峰。

```
// 中括号里面可以写驼峰可以写短横
box.innerHTML = window.getComputedStyle(img)["border-left-width"];
box.innerHTML = window.getComputedStyle(img)["borderLeftWidth"];
```

2.2 IE6,7,8 方法

不能兼容 window.getComputedStyle，IE 给我们提供了一个自己的方法。

通过对象的点语法去调用一个 currentStyle 的方法，去得到我们样式对象。

currentStyle 方法只能后面调用驼峰式的属性。不能使用 css 里面的横线写法。

也可以通过中括号来调用属性。属性名必须是驼峰的。不能用 css 的横线写法。

```
box.innerHTML = img.currentStyle["borderRightWidth"];
```

因为有兼容问题，我们需要自己封装一个函数，解决兼容性。

能力检测解决兼容问题。

```
//能力检测，如果能使用这个方法，就执行后面的分支
if(window.getComputedStyle){
    divs.innerHTML = window.getComputedStyle(image)["width"];
}else if(image.currentStyle){
    divs.innerHTML = image.currentStyle["width"];
}
```

想自己封装一个能力检测函数。

三、定时器

定时器 setInterval 也是 window 对象的一个特殊方法，每隔一段时间执行一次函数。

```
window.setInterval(函数,时间)  //这个时间用毫秒表示 1000ms = 1s。不写单位
```

```
var a = 0;
window.setInterval(function(){
    a ++;
    console.log(a);
},1000);
```

时间间隔用毫秒表示，并且不写单位。

第一个参数书写匿名函数，或者函数名。

```
var box = document.getElementById("box");
// 信号量
var nowLeft = 30;
// 定时器
setInterval(move,1000);
function move(){
    nowLeft += 10;
    if(nowLeft > 1200){
        nowLeft = 30;
    }
    box.style.left = nowLeft + "px";
}
```

setInterval 的时间间隔越小运动就越快。

函数执行几种方法：

①直接书写小括号，调用函数。

②给函数绑定事件。

③定时器执行函数。

```

```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```